# Detecting Network-Wide and Router-Specific Misconfigurations Through Data Mining

Franck Le, Sihyung Lee, *Student Member, IEEE*, Tina Wong, Hyong S. Kim, and Darrell Newcomb

***Abstract*—Recent studies have shown that router misconfigurations are common and can have dramatic consequences to the operations of a network. Misconfigurations can compromise the security of an entire network or even cause global disruptions to Internet connectivity. Several solutions have been proposed. They can detect a number of problems in real configuration files. However, these solutions share a common limitation: they are based on rules which need to be known beforehand. Violations of these rules are deemed misconfigurations. As policies typically differ among networks, these approaches are limited in the scope of mistakes they can detect. In this paper, we address the problem of router misconfigurations using data mining. We apply association rules mining to the configuration files of routers across an administrative domain to discover local, network-specific policies. Deviations from these local policies are potential misconfigurations. We have evaluated our scheme on configuration files from a large state-wide network provider, a large university campus and a high-performance research network. In this evaluation, we focused on three aspects of the configurations: user accounts, interfaces and BGP sessions. User accounts specify the users that can access the router and define the authorized commands. Interfaces are the ports used by routers to connect to different networks. Each interface may support a number of services and run various routing protocols. BGP sessions are the connections with neighboring autonomous systems (AS). BGP sessions implement the routing policies which select the routes that are filtered and the ones that are advertised to the BGP neighbors. We included the routing policies in our study. The results are promising. We discovered a number of errors that were confirmed and corrected by the network administrators. These errors would have been difficult to detect with current predefined rule-based approaches.**

***Index Terms*—Association rules mining, error detection, network management, static analysis.**

## I. Introduction

CONFIGURING routers is a tedious, error-prone and complex task. Wool [1] presents a quantitative study of configuration errors in 37 firewall engines, and found that all of the firewalls contained some misconfigurations. The author concluded that "complex rule sets are apparently too difficult for administrators to manage effectively." Feamster and Balakrishnan [2] found more than 1000 errors in the router configurations of 17 networks while focusing only on one aspect of the configurations—the one related to the configuration of the Border Gateway Protocol (BGP). BGP is the most commonly deployed inter-domain routing protocol. It allows networks to connect to each other. Mahajan *et al.* [3] measured BGP configuration errors that were visible from routing updates at the Oregon RouteViews servers over the course of 21 days, and found that misconfigurations were pervasive. About 75% of all new advertised routes were erroneously announced during that time,which was a conservative estimate according to the authors. In addition to their prevalence, misconfigurations can have large impacts: in 2004, misconfigurations in network AS9121 resulted in the propagation of 100K+ routes, leading to "misdirected/lost traffic for tens of thousands of networks" [4].

Several solutions have been proposed to deal with the router misconfiguration problem [2], [5]–[8]. All but one of them compare configurations with a list of constraints or common best practices that a network ought to follow to function correctly. This approach makes the assumption that rules violations are misconfigurations, and is very effective in detecting certain types of clear-cut problems, such as checking that internal BGP speakers form a full mesh [2], verifying that all IP addresses within a network are unique [6], and determining whether referenced routing policies are actually defined [5].

However, the identification and definition of the constraints can be a challenging task. What constitutes an error sometimes depends on the network—what is an error for one network can be common practice for another. This relativism of error definition is echoed by others. Maltz *et al.* [9] studied configuration files from 31 networks, and concluded that routing design can be diverse and each network so different from another that it is not possible to classify them: the design of network routing is eclectic, "an art where many approaches might be used to try to achieve the same result." While, for routing designs, classic textbooks generally define two architectures—the enterprise and backbone architectures—2/3 of the analyzed networks "exhibited designs that were markedly different from textbook examples and from each other." The existing approach presents limitations in the scope of errors it can detect. The rules would have to be the lowest common denominator, ones that are *universally* applicable to all networks. Unless an operator works with the tool developer to define rules for the network on a case-by-case basis, the peculiarities of the network's configurations would not be considered in the error detection process.

Solutions to detect router misconfigurations fall on a spectrum. On one extreme, we can use tools that apply best common practice rules to detect known misconfigurations. On the other

Manuscript received November 28, 2006; revised June 05, 2007 and December 18, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor O. Bonaventure. First published June 27, 2008; current version published February 19, 2009. This paper was presented in part at the ACM SIGCOMM MineNet Workshop, Pisa, Italy, 15 September 2006.

F. Le, S. Lee, T. Wong, and H. S. Kim are with Carnegie Mellon University, Pittsburgh, PA 15232 USA (e-mail: franckle@cmu.edu; sihyunglee@cmu.edu; tinawong@cmu.edu; kim@ece.cmu.edu).

D. Newcomb is with Network Operations, Corporation for Education Network Initiatives in California (CENIC), Cypress, CA 90630 USA (e-mail: darrell@cenic.org).

Digital Object Identifier 10.1109/TNET.2008.925631

extreme, there is pure data mining that ignores underlying structure of router configuration commands and domain specific knowledge. We choose to be somewhere in the middle of this spectrum of the two extremes. Our approach, called Minerals, applies data mining on router configuration files across a network to infer local, network-specific policies and detect potential errors that deviate from the inferred policies.

Our method is based on data mining and is different from existing approaches which rely on an *a priori* model.

Our contributions are summarized as follows:

- First, we propose a new method to the router misconfiguration problem. While most of existing solutions depend on an *a priori* model, our proposal automatically discovers the local rules of a network by applying data mining techniques. More specifically, *Minerals* uses association rules mining. While association rule mining has traditionally been applied to discover unknown patterns in large datasets and to predict future behaviors, we use association rule mining to infer the local, network-specific policies in a network and to detect potential misconfigurations.
- Second, we describe a set of techniques to convert network configuration into a format that is suitable for association rule mining. Raw configuration files are not conducive to data mining. Each configuration file is a concatenated list of commands typed to specify the desired behavior of a router. Therefore, we suggest a *general* method to pre-process this data and represent it in a way appropriate for association rule mining.
- Finally, we have implemented our scheme and evaluated it on the configuration files from three operational networks. We found a number of errors with minimal effort from the operators. Many of the discovered misconfigurations were deemed "serious" and "concerning". The network administrators found Minerals helpful and provided positive feedback. We describe the errors in detail in Section V. We believe that a better understanding of existing errors can help in improving and designing future router configuration languages and network management tools.

We discuss related work in the next section. Section III describes our approach, Minerals. Section IV presents our implementation and Section V shows the results of our evaluation on the configuration files from three different networks. Finally, we conclude the paper with discussions and future work.

## II. Related Work

In this section, we introduce a taxonomy to classify the approaches that analyze network configurations and describe work related to Minerals in the context of this taxonomy. The taxonomy divides the approaches in two dimensions. The spatial dimension describes how many routers are studied at a time. The temporal dimension represents whether configurations are analyzed from a specific point in time or across time.

*SR* (Single Router) explores one router's configuration from a specific time. The Router Audit Tool (RAT) [7] checks a router's configuration file against security recommendations set forth by the National Security Agency (NSA) [10], highlights the differences and assigns a score to indicate the security level of the router in question.

*MR* (Multi Router) studies multiple (or all) routers of a network from a specific time. Maltz *et al.* [9] developed a method to reverse-engineer a network's routing design from its router configuration files. The main goal is to understand the ways operational networks deploy routing protocols. Xie *et al.* [11] extracted information from configuration files to conduct static reachability analysis which can detect logic or design errors. Netsys [15] was among the first to propose misconfiguration detection through the parsing and analysis of router configuration files across a network. Feldmann and Rexford [5] adopted a similar approach but suggested to take advantage of the network's homogeneity—in terms of configuration commands—and focus on the most frequently used ones to reduce the complexity of the tool and to stay up-to-date with recent introduced features. rcc [2] concentrates on BGP misconfigurations across a network. Errors that it can detect include various BGP signaling problems and commands referring to undefined policies. All [2], [5], [15] rely on rules that need to be known *a priori* and to be provided in advance. These approaches can also detect violations of local, network-specific, policies. However, in order to achieve this, [2] and [5] require the local policies to be codified into the tools. Such customization has to be performed per network and can require significant effort from the network administrators. With EDGE, Caldwell *et al.* [6] argue that manual configuration of routers is error-prone and should be replaced by an automated process with inventory control. It suggests using data mining to mine configurations for errors but describes no details. The work of El-Arini and Killourhy [8] is perhaps the closest to ours. They use a set of Bayesian-based algorithms to detect statistical anomalies in router configurations. While the goal is similar, the approaches differ. Their algorithms focus on the frequencies of the command lines and associated attributes but do not consider the context of the commands. The authors assume that each command line is independent. However, such assumption does not hold for router configurations. Instead, Feldmann and Rexford [4] showed that there exist strong dependencies within a router and across routers of a network. In network configuration, the context often determines the values of the attributes. As an illustration, the command "`ip cdp`" may be associated with the argument of `1` (i.e., Cisco Discovery Protocol may be enabled) on internal interfaces but with the value of `0` (i.e., disabled) on external interfaces as recommended by the NSA [10]. In Minerals, we rely on the context of the configurations to infer local network policies and identify the violations. Another related work worth mentioning is the one from Engler *et al.* [16]. It automates the discovery of programming rules to then identify programming errors in source codes. While the goals and the approach share similarities, because of the differences in the application areas (programming bugs versus network misconfigurations), the means to extract the rules differs significantly.

*SR-TS* (Single Router Time Series) looks at one router's configuration across time, and *MR-TS* (Multi Router Time Series) analyzes multiple routers across time. EDGE suggests automating the provisioning tasks of a network by studying a network's configuration over time to identify recurring steps, but provides no concrete method. To the best of our knowledge, there are no other proposals in the *TS* area.

## III. Minerals

Data mining searches for patterns and rules in large data sets. Recently, it has been applied to systems and networking problems such as isolating bugs in software and detecting anomalies in traffic. Data mining can be used to identify errors in network configurations as well. Policies are usually applied across a network and evident in most routers in a network. As such, network elements often share common configurations. For example, to prevent various denial-of-service attacks, all routers or all edge routers in a network commonly implement some specific packet filters. As other illustrations of patterns in local policies, in one of the networks we have worked with, the Open Shortest Path First (OSPF) protocol was deployed and one local policy consists in having all interfaces in `passive` mode by default and explicitly use the command `no passive-interface` for backbone interfaces. This policy is to prevent users from injecting routes into the OSPF domain. Similarly, the network is deploying the Routing Information Protocol (RIP). The network has a number of subnets with user devices using RIP to learn their default paths. On non-backbone interfaces, the local policy mandates a `distribute-list out` command so that only the default route would be sent out, and a `distribute-list in` command to ensure that routers do not accept announcements originated by user devices. In this network, all backbone interfaces and all non-backbone interfaces running OSPF (or RIP) share commonalities.

Minerals uses data mining to discover local rules of a network and detects potential misconfigurations that deviate from these rules. In the networks above, an edge router lacking a required packet filter, a non-backbone interface running OSPF in a non-passive mode, or a non-backbone interface running RIP with missing `distribute-list` would all constitute violations of local policies and represent misconfigurations.

Local rules of a network can be complex and usually not captured by universal rules set forth by common best practice documents. To illustrate it, we describe the use of MD5 security to protect BGP sessions in a regional network provider and a large university campus. The "textbook" rule is to turn on MD5 if the routers involved support it. Indeed, this is the local rule of the network provider. However, the stability of MD5 implementations varies across vendors and operating system versions. Some autonomous systems (ASes) resist turning on MD5 because it delays session re-establishment after a reset. The local rules in the case of the university network are MD5 is "off" if the end point routers have exhibited problems in the past, "off" between certain ASes who preferred not to use it, "on" if the routers are from a particular vendor, and "on" on the rest of the sessions.

In this section, we describe using association rules mining [12] to find patterns of correlation between elements in router configurations across a network—outliers to the discovered patterns are potential misconfigurations. The approach assumes that there exists common configurations across routers, and the number of properly configured functions is large when compared to the number of their misconfigured counterparts. The obvious drawback of this assumption is that nonconforming but valid configurations can be classified as errors. To handle this drawback, we discuss incorporating operators' feedback to train and to improve accuracy of our technique. Note that logic or design errors which violate general network goals, such as incorrect packet filters preventing subnets from communicating, cannot be detected by Minerals.

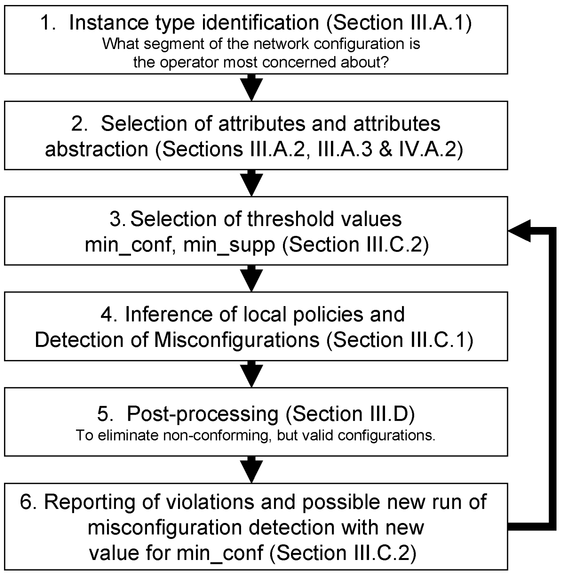
1. Instance type identification (Section III.A.1)
What segment of the network configuration is the operator most concerned about?
2. Selection of attributes and attributes abstraction (Sections III.A.2, III.A.3 & IV.A.2)
3. Selection of threshold values min_conf, min_supp (Section III.C.2)
4. Inference of local policies and Detection of Misconfigurations (Section III.C.1)
5. Post-processing (Section III.D)
To eliminate non-conforming, but valid configurations.
6. Reporting of violations and possible new run of misconfiguration detection with new value for min_conf (Section III.C.2)

Fig. 1. Overview of Minerals.

Section III-A presents a general method to preprocess routers configuration's information. Independently of the selected data mining technique, the data needs to be preprocessed since raw configuration files do not have much structure and are not conducive to data mining analysis. Section III-B provides a brief background on association rule mining. Section III-C explains how we apply association rules mining in Minerals. Finally, Section III-D describes the post-processing steps; these operations apply domain knowledge information to improve the accuracy of Minerals and reduce the number of false positives. Fig. 1 provides an overview of Minerals' main steps.

### *A. Preprocessing the Input Data*

Routers' configuration files are lists of commands and arguments that specify the routers' behaviors. These files include the interfaces, the routing protocols, the remote access and all other aspects of a router's operations. However, those files have little structure and are not fit for data mining. This section describes a general method to process the data—using domain knowledge—into a representation for data mining.

*1) Instance Type Identification:* Data mining techniques generally work on sets of instances. Each instance is characterized by a list of attributes. In Minerals, an instance type also represents a unit of error detection. Considering the local policies described above, the application of association rules mining may have allowed to discover that in that network, 99% of the non-backbone interfaces running RIP have a specific `distribute` `list in`. Each instance is an interface. Interfaces are characterized by a number of attributes including the routing protocols they run and the filters they apply. The few instances that violate the discovered rule are highlighted as potential misconfigurations. Since we focus on interface as the instance type, each error is a misconfigured interface.

We propose to divide the information in the network configuration into different instance types. Possible instance types include router, user account, interface, and BGP session.

*2) Attribute Selection:* Even though an instance type is represented by a list of attributes, not all attributes are important and suitable for data mining. Some attributes (e.g., description strings) do not contribute to the operation of a network. We suggest to remove them since errors in those fields do not impact the network's operations. Other attributes may be modified frequently as part of the day-to-day operations and may not be conducive to data mining either. For instance, operators may tune the OSPF weights to avoid link congestions [17].

Instead, we select attributes for each instance type so that only relevant information is retained. As an example, an interface can be characterized by the running routing protocols, the supported services, and the applied packet filters.

*3) Attribute Abstraction:* The type (e.g., Boolean, integer, string, etc.) that we select to represent the different attributes determines the rules and types of errors that will be discovered. Using domain knowledge, we propose to further process the selected attributes as follows.

- *Boolean*. Applying data mining with attributes that are expected to have different values across the instances may reveal few or no common patterns. Since we may not be interested in the exact values of these attributes but rather whether the attribute exists, we change the attribute so it takes on Boolean values. For example, considering the "user account" instance type, this instance type may have several attributes including the "password" that has been set for this account. The value of the password is not as important as the existence of an explicit, non-default password. Leaving the password string as is, and applying data mining may reveal that certain users have different passwords on the different devices. Such result may not be of interest. Instead, an account (e.g., with super-user privilege) which is lacking a required password may consist a more concerning error.
- *Group*. If an attribute is characterized by values that belong to groups and the values themselves differ across the instances, we convert the attribute so that it takes on the group names as its values. The reasoning is the same as above. For example, instead of storing the actual IP addresses of an interface, the IP addresses are classified as private or public. Similarly, the AS number of the other end point of a BGP session can be used to derive the type of a BGP session (i.e., internal or external).

*4) Data Cleaning:* Each router vendor has its independent configuration language, e.g., Cisco's IOS versus Juniper JUNOS, that can be vastly different from others. An intermediate representation is necessary to compare configurations from different vendors, which may bear no resemblance syntactically but are semantically equivalent. To illustrate, IOS relies on privilege levels (0–15) to define a user's rights whereas JUNOS uses the concept of classes (super-user, operator, etc). An IOS's privilege level 15 can be considered comparable to JUNOS's "super-user" class. However, sometimes, an intermediate representation is not sufficient. Vendors may implement features in ways that cannot be reconciled across vendors. For example, there seems to be no equivalent of IOS's privilege level 6 in JUNOS.

There can also be various ways to implement the same functionality. In IOS, a routing prefix can be filtered on a BGP session either directly through applying a prefix-list, or by using prefix-list in a route map and applying the route map on the BGP session. We can clean the data such that attributes that define the same functionality are grouped into one attribute. Finally, in the previous example, the attributes "incoming prefix-list" and "incoming route map" can also be subsumed into "incoming policies" as they both specify routing policies.

## B. Background on Association Rules Mining

This section gives a brief overview of association rules mining. Interested readers can refer to [12] for details.

We select the association rules mining technique to detect anomalies in router configurations for the following reasons.

- To be effective, network policies are usually applied on most or all objects of the same category across a network.
- Network policies can often be expressed as rules. For example, the policy "BGP sessions with external ASes must be authenticated" can be represented by the rule $\mathrm{eBGP} = 1 \Rightarrow \mathrm{MD5} = 1$.
- Deviations from local network policies can either be misconfigurations, nonconforming valid configurations, or temporary solutions, all of which should be audited periodically.

At a high level, association rules mining examines the statistical properties of correlations present in a large data set. In the traditional data-mining context, it is used to classify and predict an attribute or combination of attributes. The prediction is called a rule. Let $a, b, c, \ldots$ be attributes. Association mining searches for rules $\{a = a_i \wedge b = b_j \wedge \ldots\} \Rightarrow m = m_k$, in which any combination of unique attributes can be on the left-hand side. In other words, a rule takes the form "if $X$ then $Y$", or $X \Rightarrow Y$, where the left-hand side represents the "*condition*" and the right-hand side the "*consequence*". The pattern $\{a = a_i \wedge b = b_j \wedge \ldots\}$ is also known as an item set.

$\{\varnothing\}$ is a 0-item item set,
$\{a = a_i\}$ is a 1-item item set,
$\{a = a_i \wedge b = b_j\}$ is a 2-item item set, and so on.

Given an item set $X$, its support, $S(X)$, is defined as the number of instances that satisfy the condition $X$. Given $X \Rightarrow Y$, its confidence $C(X \Rightarrow Y)$ is calculated as $S(X \wedge Y)/S(X)$.

TABLE I
ILLUSTRATION OF ASSOCIATION RULES MINING

| INSTANCE ID | $a$ | $b$ | $c$ | ... |
|---|---|---|---|---|
| 1 | $a_1$ | $b_1$ | $c_1$ | ... |
| 2 | $a_1$ | $b_1$ | $c_1$ | ... |
| 3 | $a_1$ | $b_1$ | $c_2$ | ... |
| 4 | $a_2$ | $b_2$ | $c_1$ | ... |
| 5 | $a_2$ | $b_2$ | $c_2$ | ... |

An instance is characterized by a number of attributes (e.g., $a$, $b$, $c$, etc.).

We illustrate association mining using Table I.

$$\begin{aligned}
&S(\emptyset) = 5\\
&S(a = a_1) = 3\\
&S(a = a_2) = 2\\
&\dots\\
&S(a = a_1 \wedge b = b_1 \wedge c = c_1) = 2\\
&C(\emptyset \Rightarrow a = a_1) = 3/5 = 0.6\\
&C(a = a_1 \Rightarrow b = b_1) = 3/3 = 1\\
&C(a = a_1 \Rightarrow c = c_1) = 2/3 = 0.67\\
&\dots\\
&C(a = a_1 \wedge b = b_1 \Rightarrow c = c_1) = 2/3 = 0.67
\end{aligned}$$

In our context, rules with high confidence are considered reflections of local network policies. Instances that deviate from these rules (i.e., $X \Rightarrow \neg Y$) are identified as potential misconfigurations because they do not comply with the inferred policies.

### C. Applying Association Rules Mining in Minerals

This section describes how we apply association rules mining in the context of Minerals. It describes the algorithm to infer the network-specific local policies and to identify the potential misconfigurations.

*1) An Algorithm to Infer Local Policies and Detect Misconfigurations:* In Minerals, the generation of association rules and detection of violations from those rules consists of the following four steps.

*Step 1) Item set generation.* For each instance type, this step generates all possible combinations of item sets, e.g., all 1-item item sets, 2-item item sets, etc., with the restriction that the support is over the threshold $min\_supp$. A number of algorithms have been proposed by the data mining research community to compute item sets efficiently.

*Step 2) Inference of local policies.* The goal of this step is to generate association rules. Minerals filters out rules with confidence values lower than the threshold $min\_conf$ as we want to find rules that are the most pertinent and more likely to be reflections of local policies. This step can also apply domain-specific knowledge to eliminate irrelevant rules and thus reduce the false alarm rate: if common sense says attribute $a$ cannot imply or be correlated with $b$, we eliminate all rules of the form $\ldots \wedge a = * \wedge \ldots \Rightarrow b = *$. Even though this only needs to be done once and can be henceforth remembered by the Minerals algorithm, it nevertheless can be labor intensive.

We show in our evaluation that without applying this filtering rule, we are still able to achieve decent results.

*Step 3) Violation detection.* Since we are focusing on misconfigurations, we do not consider rules with confidence equal to 1. Violations are instances that do not comply to a rule $X \Rightarrow Y$ with $min\_conf \leq C(X \Rightarrow Y) < 1$. The attribute that might be misconfigured is either in $Y$ or $X$—usually, the rule has the expected, common value, whereas the violation has a potentially wrong value. We also eliminate rules that generate more than $max\_violations$ number of violations. The reason behind using $max\_violations$ is that the number of misconfigurations should be small in a network. If a rule results in a large number of violations, the violations are most probably not misconfigurations, and the rule unlikely to reflect a local policy. We use a simple default value of 10.

*Step 4) Filter and report.* The last step reports the identified violations in decreasing order of confidence. Since multiple rules can point to the same misconfigured instance, we only report the instance once and indicate the rule with the smallest item set size on the left hand side. Such an operation allows to focus on the attributes that raised the violations. Also, it keeps the rules simple. Experience shows that those rules are often easier to interpret.

*2) Selection of Parameters: $min\_conf$ and $min\_supp$:* The algorithm proposed in the previous section relies on a number of parameters. This section describes several methods to determine $min\_conf$ and $min\_supp$.

*Feedback-based method*: Because our approach assumes that the number of properly configured instances is high, $min\_conf$ should have a large value. It can be initialized to 0.99 and then incrementally decreased. After each run of Minerals, the network operator can provide direct feedback on whether and how a reported violation is a false alarm. The operator can indicate whether the discovered rule is or not a reflection of his network policy, and if the violation is an exception to a valid policy. This knowledge can then be fed back into subsequent runs of Minerals to filter rules and violations. The lists of rules and instances previously raised and confirmed as valid exceptions by the network operators are recorded. Subsequent results are then compared with these lists before being brought up to the attention of the operator. For example, we assume that in the first run of Minerals ($min\_confidence$ set to 0.99), a rule was discovered ($a \Rightarrow b$) and one instance, violating this rule, was reported. When presenting the results to the operator, this latter may indicate that the discovered rule is not a reflection of a local policy. Consequently, in subsequent runs of Minerals (e.g., with $min\_confidence < 0.99$), ($a \Rightarrow b$) will not be considered and instances violating it will not be reported. Alternatively, to ease the burden from the operator, Minerals can learn indirectly. Violations that are reported but not corrected over time are assumed to be exceptions.

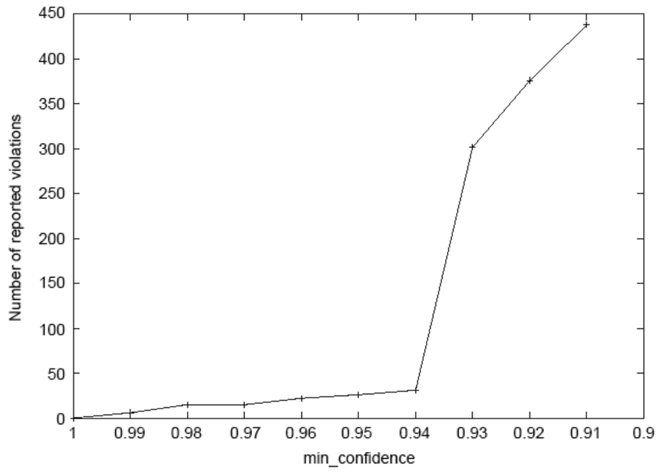

Fig. 2. Evolution of number of reported violations (after post-processing) as $min\_conf$ is decreased. In this case, the $min\_conf$ is chosen to be 0.94.

*Heuristic-based method*: As an alternative, we can decide of the value of $min\_conf$, based on the evolution of the number of reported violations as we decrease possible $min\_conf$ candidate values. For each value of $min\_conf$, we apply the algorithms above to infer the local network-specific policies and then, we identify the instances that violate them. We apply a list of post-processing rules (described in Section III-D) to eliminate several false alarms. As such, for each value of $min\_conf$, we obtain a number of violations. We plot these numbers of reported violations for possible $min\_conf$ values as in Fig. 2. A guideline reduces $min\_conf$ until we find the knee of the curve, i.e., the point at which there is a noticeable increase in the number of violations afterwards. We use the value of $min\_conf$ immediately before the knee of the curve. Assuming the evolution depicted in Fig. 1, we set $min\_conf$ to 0.94 and less than 40 violations are reported. The preferable value is determined from the following reasoning: if the number of reported violations is large, those instances are most probably not misconfigurations. It is important to note that each derived value of $min\_conf$ is specific to a dataset. As an example, we may find one value of $min\_conf$ to apply when running Minerals on the interfaces aspect of a network, and we may find a different value for $min\_conf$ to be used on the routers aspect of that network.

Both the feedback-based and heuristic-based methods can be applied together. The heuristic-based technique can first be applied. Then, the feedback-based method can be used to gradually decrease the $min\_conf$ values. Such approaches allow to discover errors while keeping the false positives and the number of analyses low.

Finally, for $min\_supp$, setting it to a low value may result in an unnecessary large number of frequent item sets. Setting it to a large value may prevent the detection of potential misconfigurations. We set $min\_supp = \min\{k \in N^* | k \geq (1 - min\_conf)^{-1}\}$ where $N^*$ is the set of integers except the null value.

## D. Post-Processing

After pre-processing the configurations and running our association mining algorithm on them, we apply a post-processing step to filter the results. The goal is to reduce false alarms by demoting violations that are highly likely to be nonconforming, but valid configurations.

The first post-processing step is related to routing policies and attempts to identify whether the policies for a group of neighbor routers in the same AS are nonconforming but intentional. Even though policies for a specific neighbor AS are not consistent with policies with other ASes peering with a network, they may not be misconfigurations because it is not uncommon for a network to customize policies for each neighboring AS. We assume that if policies are consistent across all BGP sessions to the same AS, the violation highlighted by Minerals is likely not a misconfiguration. This post-processing works as follows.

We note $s$, a BGP session, and $s.attribute\_name$, the value of $attribute\_name$ in $s$ (e.g., $s.AS\_number$ is the AS number associated with the external end of $s$). We use a Boolean variable, $\mathrm{demotable}$, to record the result.

```
**for each** BGP session $s_i$ violating an inferred rule
$(a = a_0 \wedge \ldots \wedge b = b_0) \Rightarrow m = m_0$
    $\mathrm{demotable} = 1;$
    **for each** $s_j$ **s.t.** i $\neq$ j and $s_i.AS\_number = s_i.AS\_number$ and $s_i.peer\_group = s_i.peer\_group$
        **if** $(s_i.m \neq s_i.m)$ {$\mathrm{demotable} = 0$; break;}
    **end for**
    // if all BGP sessions having the same AS and
    // peer_group as $s_i$ present the same value for
    // the attribute $m$, demote $s_i$.
    **if** $(\mathrm{demotable} = 1)$ **demote** $s_i$;
**end for**
```

Note that the attributes $a, b, \ldots, m$ above do not necessarily represent the complete routing policy applied to a BGP session. In Minerals, a rule can be a subset of a routing policy. We do not assume all routing policies to the same AS must be identical. In fact, in some configurations, each router can add a distinct community tag, thus rendering every route map unique if we compare route map in its entirety. Minerals does not highlight each of these route maps as an anomaly, because it only focuses on commands that appear frequently in routing policies.

There are several ways to demote a violation. Currently, we simply discard the violation. However, this may cause errors to go undetected. A misconfiguration can be present in all BGP sessions to the same AS because of a cut-and-paste operation. There is an inherent tradeoff between the detection and false alarm rates of Minerals. Several operators have told us that they want to know about all inconsistencies in their configurations, even though some are highly likely to be valid. Another way to demote a violation is to lower its priority when presenting the results to the network operator.

The second post-processing step is related to MD5 security. Best common practice says it should be enabled on eBGP sessions. However, in practice, some older routers do not support this feature well and operators sometimes disable it for performance or stability reasons. If Minerals reports a violation of missing MD5 protection on a BGP session, this post-processing step checks to see if any of the two following conditions is satisfied: i) the offending router is having sessions with other ASes that require MD5, but all sessions from this router have MD5 disabled, or ii) all sessions to the other end point of the session have MD5 disabled. If so, we assume the local policy says to disable MD5 on this router or the other end point, and demote the violation.

The last post-processing step identifies simple BGP sessions, which we define as sessions which either use default routing, or block all incoming or outgoing route advertisements. If Minerals raises a violation on a simple BGP session, we demote it.

## IV. Implementation and Evaluation

We implemented and evaluated Minerals on the configuration files from three networks (see Table III).

### A. *Case Study of Accounts, Interfaces, BGP Sessions,and Their Associated Routing Policies*

Many instance types can be identified and mined for misconfigurations. We conducted a study on three aspects of configurations, or what we call instance types: user account, interface and BGP session. For the BGP session, we also considered the routing policies. We concentrated on these aspects since we believe they are prone to mistakes. The results presented in Section V confirmed that these aspects of the configuration contain errors.

Below, we describe how we select or generate attributes for each of the three instance types. For user account, interface and BGP session, the attributes are the same for any network. The attributes represent information that does not vary much across different networks. On the other hand, for routing policies, the attributes are different for each network evaluated. The attributes are extracted from the configurations based on the details of the commands and the action taken by the routing policies. Thus, the attributes are specific for a network, and the number of attributes is dependent on the structure and complexity of the routing policies within that network. For aspects of configurations that are highly customized according to the needs of a network, generating attributes in this way would yield more local, network-specific rules. As shown in this case study, the two ways of choosing attributes are both useful in the detection of misconfigurations.

*1) Selecting Attributes for Account, Interface, and BGP Session:* User account is identified by the (router name, user name) pair and is characterized by the attributes "password" and assigned "privilege level." For the interface instance type, there are a myriad of configuration options associated with it. We avoid attributes that are functionally independent thus not conducive to data mining. We select an interface's IP address and whether it serves as a loopback. Finally, a BGP session is identified by the (router name, neighbor router) pair. We select the following attributes, which characterize a BGP session at a high-level: the AS number of the neighbor router, incoming routing policies, outgoing routing policies and authentication scheme. Routing policies are usually considered complex and prone to errors. As such, we decide to analyze them in greater detail. The following section describes the representation we adopt for the routing policies applied to each BGP session.

TABLE II
Summary of Preprocessing Outcome

| Instance Type | Attribute | Value |
|---|---|---|
| Account | username | string |
| | password | bool |
| | privilege | int |
| BGP session | type | internal/external |
| | MD5 | bool |
| | incoming policies | bool |
| | outgoing policies | bool |
| | AS number | int |
| Interface | loopback | bool |
| | IP address | bool |
| | IP address type | private/public |

This table represents the format of some of the attributes characterizing diverse instance types. The final table that is considered for association rules mining contains a significantly larger number of attributes. More specifically, the addition of the attributes related to routing policies resulted in the addition of more than 800 attributes in certain networks.

The results of the initial phase of the preprocessing for the three selected instance types are summarized in Table II. They serve as input to the association rules mining algorithm described in the previous sections. Additional attributes can be added to characterize the different instance types. For example, routing protocols and services (proxy-arp, Cisco Discovery Protocol, etc.) can be attributes of the interfaces.

*2) Representing Routing Policies:* This section focuses on routing policies. We describe how we represent routing policies so that association rules mining can be applied to them and misconfigurations, detected. Routing policies are usually considered complex to configure and their structure, intricate.

*a) Background on routing policies:* Routing policies control which routes a router or a network accepts, filters and forwards. At a high level, routing policies allow a network to specify the flow of its inbound and outbound traffic. Redistribution of routes between routing processes (e.g., from OSPF to BGP, from RIP to OSPF) can also be configured.

Each router vendor has its own proprietary configuration language. Some configuration features are vendor-specific and not available in other vendors. In the rest of this paper, we use Cisco IOS terminology and syntax, but both our methodology and implementation work for Cisco IOS and Juniper JUNOS. Fig. 3 is a factitious excerpt of a router configuration file. We use it to illustrate how routing protocols and policies are defined.

Lines 100 to 110 specify the interfaces of the router.

Lines 200–202 configure the RIP routing process. The `network` command indicates that all interfaces in `192.168.1.0` are to use RIP. The command with the keyword `distribute-list out` restricts advertised updates onto Ethernet0/0 according to the access list called 1, defined in

```
100 interface Loopback0
101  ip address 10.10.10.1 255.255.255.255
102  no ip redirects
103  no ip directed-broadcast
104  ip pim sparse-mode
105 !
106 interface Ethernet0/0
107  ip address 192.168.1.1 255.255.255.0
108  no ip directed-broadcast
109  ip route-cache cef
110 !
...
200 router rip
201  network 192.168.1.0
202  distribute-list 1 out Ethernet0/0
...
300 router bgp 100
301  neighbor dora peer-group
302  neighbor dora remote-as 200
303  neighbor 4.5.6.1 peer-group dora
304  neighbor dora prefix-list pf_dora in
305  neighbor dora route-map from_dora in
306  neighbor dora route-map to_dora out
...
400 ip community-list 2 permit 100:4
401 ip prefix-list pf_dora permit 10.10.0.0/16
402 ip prefix-list pf_dora permit 10.11.0.0/16
403 access-list 1 permit 0.0.0.0
404 access-list 1 deny any
...
500 route-map from_dora permit 100
501  set local-preference 100
502  set community 100:1 100:2 100:3
503
504 route-map to_dora deny 10
505  match community 2
```

Fig. 3. Excerpt of a Cisco router configuration file.

lines 403–404. The access list permits only a default route to be advertised.

Lines 300–306 configure the BGP routing process. The number `100` is the local autonomous system (AS) number. In line 301, the command `peer-group` creates a group called `dora`. `peer-group` facilitates the application and modification of routing policies on a set of neighbors. Line 302 associates the AS number `200` to `dora`, which tells us whether the BGP session is with external or internal neighbors. Line 303 assigns the BGP neighbor with IP address `4.5.6.1` to the peer group `dora`. Line 304 applies a prefix list called `pf_dora` to all BGP sessions of `dora`. `pf_dora` is defined in lines 401–402, and permits only the prefixes `10.10.0.0/16` and `10.11.0.0/16`. The keyword `in` specifies that the prefix list is applied to incoming advertisements. Therefore, only `10.10.0.0/16` and `10.11.0.0/16` are accepted from the peer group `dora`.

Lines 305–306 apply the import policy `from_dora` and export policy `to_dora` to the BGP sessions in `dora`, which are defined in lines 500–502 and lines 504–505, respectively. `from_dora` accepts all routes and assigns them a local preference value of `100`. It also tags the routes with three communities `100:1, 100:2` and `100:3. to_dora` denies routes which carry the community tag listed in the community-list `2`, defined in line 400. Routes marked with `100:4` are not advertised to BGP neighbors of `dora`.

*b) A representation for routing policies:* There are many commands to configure routing policies. We choose a small subset of them based on our observation that a significant portion of routing policies is related to BGP communities, AS Path, prefix lists and local preference. We distill these commands to remove vendor-specific syntax. After parsing the router configurations from vendor-specific syntax to a common intermediate representation, we expand each reference (i.e., prefix-list, community-list, and aspath-list), to the list definition. This allows us to focus on the values of the lists and not the names assigned to the lists. For example, `sanity` is used to name a prefix-list containing unallocated IP address space in one router but the name `bogon` is used in another router to specify the same prefix-list.

We then distill each action of a routing policy into an attribute. For each line of a routing policy applied to a BGP session, we extract the direction of the policy (incoming or outgoing), the action taken (permit or deny, set community or local preference) and the conditions being matched (community, AS path, prefix-list). For example, the routing policies defined in Fig. 1 and applied to the BGP sessions with `dora` would be converted into the following five attributes:

```
in_permit_set_localpref_100
in_permit_set_comm_100_1
in_permit_set_comm_100_2
in_permit_set_comm_100_3
out_deny_match_comm_100_4
```

An attribute can take on different values. In our case, an attribute takes on a Boolean value of either 1 or 0: 1 if a routing policy contains the action represented by the attribute, and 0 otherwise. Thus, the attributes above would take on the value 1 for the BGP sessions where the routing policy is applied on.

As such, each routing policy is converted into a number of attributes and these attributes are added to the BGP session instance type. The number of attributes related to routing policies varies per network. Networks with many and complex policies will tend to have more attributes.

In our current implementation, the pre-processing step only yields a partial semantic comparison of routing policies. Regular expressions are not expanded. Two regular expressions may differ syntactically but have the same semantics, e.g., (100|200) and (1–2)00. Vendor-specific command syntax also presents additional challenges. For example, to specify an AS path that includes the AS number 100, IOS would define it as _100_, where as JUNOS would define it as ".* 100 .*". Finally, our representation does not consider the ordering of lines within a routing policy. Just like a program can be written in many ways, one can achieve the same routing policy with different configurations.

Despite these limitations in our current implementation, we believe that the adopted representation is acceptable for the following reasons. First, to keep problems to a minimum, networks are commonly configured by a small group of highly skilled experts. Second, existing configurations are often used as templates to configure another router or another routing policy. There is a substantial amount of "cut-and-paste" in the three networks we studied. Later, we show that Minerals found a number of errors in the routing policies' aspect which are confirmed by the network operators.

TABLE III
SUMMARY OF DATASET USED IN EVALUATION

| Network name | Num. of routers total (IOS, JUNOS) | Number of instances | | |
|---|---|---|---|---|
| | | Account | Interface | BGP |
| cenicDC | 44 (37, 7) | 204 | 831 | 1958 |
| cenicHPR | 6 (6, 0) | 8 | 102 | 110 |
| UCB | 67 (65, 2) | 67 | 1507 | 191 |

CENIC is the California state-wide network service provider for education and research institutions: DC is a backbone network serving education users, and HPR is a high-performance research network providing advanced services for large application users. UCB is the UC Berkeley network.

## B. Dataset

Our dataset comprises configuration files from the CENIC CalREN-DC and CalREN-HPR networks,[1] and the UC Berkeley campus network. Information related to each network is summarized in Table III. Two of the networks include both Juniper and Cisco routers and our study considered both types of routers. For each of the three networks, we analyzed a particular snapshot of its configuration files, taken between January and April 2006.

Table III describes the number of instances we found in each network, after parsing the router configuration files. In addition to the five attributes of the BGP instance type (type, MD5, incoming policies, outgoing policies, AS number) described in Table II, the processing of the routing policies (Section IV-A2.b) adds a number of attributes. The number of additional attributes varies per network and increases as the number of distinct routing policy actions gets larger. These pre-processing steps generated 70 attributes in UCB, 177 in cenicHPR, and 761 in cenicDC. In our dataset, cenicDC has the largest number of attributes. This is not surprising since as a service provider, it needs to define different routing policies for its provider(s), peers, and other neighbors. The policies are indeed complex. The BGP documentation of CENIC lists 390 communities. This network also applies a large number of prefix-lists to prevent accidental address leaks, and applies routing policies for traffic engineering purposes.

## C. Implementation and Performance

Table IV lists the components of Minerals and their corresponding programming languages. To generate the frequent item sets, we use the *LCM ver. 2* algorithm [13]. This algorithm actually outputs closed frequent item sets. The closed frequent item set representation significantly reduces the space requirements without affecting our results. In order to apply the *LCM ver. 2* algorithm, we needed to convert the input attributes into Boolean format.

The aspect that requires the most time is the pre-processing. CenicDC presents the largest observed time. It is in part due to the large number of BGP sessions with routing policies. Also, adding a new attribute to a large table can require several seconds in MySQL. In the dataset from CenicDC, the addition of a new attribute into the table representing the BGP sessions can take more than 6 seconds.

[1]Corporation for Education Network Initiatives in California (CENIC). http://www.cenic.org/

TABLE IV
IMPLEMENTATION AND PERFORMANCE CHARACTERISTICS

| Component | Alg/Prog. language | Elapsed Time[1] | | |
|---|---|---|---|---|
| | | cenicHPR | cenicDC | UCB |
| Parser | Lex & Yacc | 00:17.47 | 01:41.13 | 00:11.61 |
| Preprocess | C | 00:14.57 | 03:44.30 | 00:58.94 |
| Closed FIM[2] | LCM ver. 2 | < 00:00.01 | 00:00.01 | < 00:00.01 |
| Assoc. rules & Post-process[2] | C | 00:02.21 | 02:06.50 | 00:00.05 |

[1] Elapsed time in minutes:seconds. The times were observed when running the components on a machine with a CPU of 2.8 GHz and 2 Gb of memory.
[2] Elapsed times for the "Closed FIM" and "Assoc. rules & Post-process" correspond to the observed time when running these functions on the BGP session instance type (which has the largest number of attributes)

However, the total processing times (including the parsing, preprocessing, discovery of local rules and detection of misconfigurations) still remain in the order of 1 to 10 minutes for the three networks we analyzed.

## D. Usage Complexity

For a system to be practical and useful in operational settings, it must be easy to configure and run. We believe Minerals satisfies these requirements. The configuration files need to be placed in a common directory. The code processes them without any further manual intervention from the operators. For a value of $min\_confidence$, the running time varies from one to a few minutes depending on the size of the configurations. After the code parses the configuration files, pre-processes the extracted information, infers the local policies, identifies the violations, and applies the post-processing rules, the operators are presented with the results, i.e., the anomalies and the policies that are violated by these instances. For all three networks, by looking at the list of rules and violations, the operators were able to quickly tell whether a raised instance was a misconfiguration and needed action. Rarely did the operators need to go into the configuration files to collect more information regarding the highlighted anomalies.

If the operator wants to determine an "optimal" $min\_confidence$ value before looking at the results (e.g., through the heuristic-based method presented above), it first needs to plot the evolution of the number of reported errors over different $min\_confidence$ values. This can be simply executed since the operator only needs to input a varying $min\_confidence$ value and the code will indicate the corresponding number of violations. This step can be automated.

Finally, a frequently raised concern with association rule mining is the large number of rules that can be generated. Minerals addresses this issue through the use of closed frequent item sets (Section IV-C) and the reporting of potentially misconfigured instances (Section III-C) instead of the discovered rules. These two means remove existing redundancy and maintain the size of the output to a reasonable size, as illustrated in Table V.

TABLE V
SUMMARY OF MINERALS' RESULTS

| Instance type | Needs fix | False alarms |
|---|---|---|
| Account *min_conf = 0.90* | 9 | 6 |
| Interface *min_conf = 0.90* | 10 | 0 |
| BGP *min_conf = [0.90, 0.97]* | 20 | 17 |

$min\_conf$ is determined through the heuristic-based approach. Minerals is first applied to the datasets using these $min\_conf$ values. The presented numbers (Needs fix, false positive) are the results after this initial phase. In a second phase, feedback from the operators is included to discover additional errors while discarding nonconforming but valid configurations.

## V. RESULTS

We set $min\_conf = \min\{0.90, k\}$ where $k$ is the value obtained through the heuristic-based method described in Section III-C. The reason we used a "$min$" is to ensure that $min\_conf$ keeps a high value. This is motivated by the fact that we infer local policies from discovered rules with a high confidence value. The obtained values ranged from 0.97 to 0.90 depending on the instance type. Since we do not expect $min\_conf$ to go lower than 0.90, we fixed the value of $min\_supp$ to 10. We did not pre-filter any association rules to allow the algorithm to find all possible policies and violations.

Table V summarizes the results in two categories: *needs fix* and *false alarms*. *Needs fix* is composed of two categories: i) misconfigurations that are confirmed by the network operators, and ii) transitions, i.e., violations that reflect configurations (e.g., policies) undergoing change. While such violations are not always errors, they should be brought to the attention of network operators. If not updated in a timely manner, these cases can become errors. For example, in one case, an outdated community that was not removed led to a customer filtering out an intended route. *False alarms* are violations raised by Minerals that are nonconforming but correct configurations. Interestingly, the operators were interested in knowing about these false positives as well. Those instances are outliers and it is important to keep track of these configurations as the network evolves.

We describe the results in more detail. All results are anonymized. First, we determined the $min\_conf$ values through the heuristic-based method. Section V-A presents the errors that were detected in this initial phase and confirmed by the operators. Then, in the second phase, we re-ran Minerals with lower $min\_conf$ values (e.g., $min\_conf$ lowered from 0.97 to 0.95). When processing the new violations, we incorporate the feedback from the operators from the previous phase. This process allows us to detect additional errors while keeping the false positives to a low value. Section V-B describes the additional error types that were discovered in this second phase. Finally, Section V-C explains the false positives.

*Summary of discovered error types:*

- A number of user accounts with super-user privileges are missing password.
- A few interfaces, used for bootstrapping purposes, were detected. They should have been removed after the initial deployment phase.

A number of errors were discovered in the routing policies applied to the BGP sessions. More specifically:

- A significant fraction of the errors related to the configuration of routing policies are missing communities. These misconfigurations affected the redistribution of routes, the aggregation of prefixes and the accounting of certain traffic.
- Typos were found in the definitions of communities. These errors can prevent the intended actions to be performed.
- A number of sessions lacked import or export policy.
- The detected errors also included missing prefix-lists, both in the incoming and outgoing directions.
- Sessions lacked MD5 security violating local policies.

### A. Confirmed "Needs Fix" in First Phase

*Missing passwords.* In one network, Minerals found super-user accounts without passwords. The inferred local rule was that account with super-user privileges must have a password. Minerals reported the instances that violated this rule. The operator confirmed those accounts as errors, which presented a security breach.

Universal rules cannot capture local policies regarding the user accounts in different networks. Some networks may allow external parties to access certain information (e.g., routing tables) without any password. Other networks may consider the same type of information to be confidential and mandate a password to access them. The starting privilege level where a network may require the presence of a password is network dependent. Minerals can infer these local policies.

*Unused interfaces.* In one network, 95% of the interfaces had public IP addresses, and Minerals highlighted interfaces using private IP addresses as violations. According to the operator, private IP addresses are used to bootstrap routers at the beginning of deployment, and should be deleted afterwards. These confirmed misconfigurations were corrected by the operator.

It is difficult for rule-based approaches that rely on universal rules to detect such errors, because the presence of private IP addresses does not automatically imply a misconfiguration. Some networks use private IP addresses permanently, e.g., for network management devices. The advantage of data mining is it considers the statistical properties of a network, and in this case, reveals the outlier interfaces as errors.

*Missing MD5 security.* Minerals found a large number of violations on missing MD5 security. The second post-processing step described in III.D removed a substantial portion of these violations. In one case, the violations are sessions to ASes that preferred not to use MD5. All the sessions to those ASes are found with MD5 disabled and therefore removed. Another case involves a specific router. The post-processing step found that MD5 is disabled on every session involving that router. It is likely that router does not support MD5. It turns out some ASes resist turning on MD5 because it adds delays to session re-establishment after a reset.

The rest of the violations are sessions with neighboring networks that require MD5. These sessions were not removed by the post-processing step, and confirmed as errors.

*Omitted export policies to external neighbors.* In one network, eBGP sessions that applied import policies also had export policies, except in few sessions. The operator confirmed that these sessions were errors and described them as "very concerning." The absence of export policies can result in a list of undesired effects, such as unintentional transit service.

*Missing incoming prefix-list.* A local network policy requires a prefix-list to be applied on all incoming external BGP sessions. The prefix-list turns out to be a sanity check that discards route announcements to private IP address. Few BGP sessions in this network violate this policy. One version of a correct route map consists of the following:

```
route map from_abc permit 50
match ip address prefix-list sanity
match community...
set local-preference 90
set comm-list...delete
set community...additive
```

One version of the violations is as follows:

```
route map from_bcd permit 50
set local-preference 100
set comm-list...delete
set community...additive
```

Although the two route maps above are different in several places—different local preference values, match community statement in the first but not the second—Minerals is able to detect similarities in parts of the route maps and points out the missing prefix-list statement. As mentioned before, Minerals do not require routing policies to be identical in order to find inconsistencies because association rules mining can work on subsets of attributes defining a policy.

*Missing outgoing prefix-list.* Similar to the above, a local network policy requires a prefix-list to be applied on all outgoing external BGP sessions. The prefix-list is also a sanity check. In IOS, a `match ip address prefix-list sanity` is used. A number of BGP sessions are missing this statement.

*Missing communities.* 45% (9/20) of all the confirmed errors related to routing policies are related to communities. A number of these errors are accidental omission of community tagging, which lead to several undesired outcomes. In two cases, missing communities prevent routes from being advertised to a set of peers. Since these two peers each have two eBGP sessions to the network and the absence of community is only affecting one eBGP session, routes are still advertised to the peers. However, if the BGP session is to be temporarily disabled (e.g., for maintenance purposes), such misconfiguration will affect the connectivity of the peers.

In another instance, one eBGP session was lacking a community $xyz$ which prevented the intended routes to be advertised. Further analysis revealed that the presence of another community $abc$ was advertising the desired routes. The operator explained that the network was going through a routing policy change. The tagging of routes coming from commodity peers was modified to allow more flexible customizations. However, sessions that are not updated in a timely manner can lead to undesired effects. As an example, a neighbor may be added and its routes may be tagged with community $abc$ to be re-advertised to a set of neighbors. Sessions still relying on the old community $xyz$ may not receive the newly added routes.

*Incorrect community-list definitions.* In a number of routers, a community-list is incorrectly defined.

```
ip community-list 50 permit permit
regexp
...
```

The repetition of the word "permit" is an error. While a syntax-based configuration checker can also detect this type of error with a specific rule looking out for "permit permit", Minerals allow us to detect syntactical abnormalities using the same algorithm. In this case, a large number of BGP sessions have this error and a few sessions have the intended definitions. Minerals highlighted the errors as the rule and the violations are actually the correct configurations.

```
ip community-list 60 permit regexp
...
```

As a consequence of this error, routes that are supposed to be matched (i.e., routes that carry communities matching `regexp`) and that should receive a set of actions, will not. This is because the community-list 50 will only match routes that include both communities `permit` and `regexp`. However, there is no community with the value `permit`.

*Inconsistent community-list definition.* A community-list was defined in the same way in 18 routers, but was slightly different in another router. It turned out that the definition is being updated but the modification is not complete when we carried out our evaluation.

Common definition (present in 18 routers)

```
ip community-list 190 permit
100:6550[1-9]
ip community-list 190 permit
100:6551[0-4]
```

Outlier (present in one router)

```
ip community-list 190 permit
100:6550[1-9]
ip community-list 190 permit
100:6551[0-9]
```

The outlier in this case is the intended definition, thus all other definitions should be updated. As a consequence of this error, routes that include communities `100:65515`, `100:65516`, `100:65517`, `100:65518`, or `100:65519` will not receive the intended set of actions in the misconfigured routers.

## B. Confirmed "Needs Fix" in Second Phase

In the second phase, we gradually decrease the value of $min\_conf$ of the datasets where $min\_conf$ was high (e.g., 0.97). We analyze the additional violations and we take into consideration the previous feedback from the operators to discard the false positives.

A number of errors consist of violations of rules previously described (e.g., missing prefix-list, missing MD5, etc.)

Additionally, we detected the following new error types.

*Missing communities*: We discovered additional missing communities with diverse consequences. In a case, prefixes to be aggregated are to be tagged with specific communities and filtered out at the egress routers. The absence of the communities on certain routes affected route aggregation and resulted in the advertisement of some of the more specific prefixes. One interesting error impacted accounting. As communities control route announcement, a missing community prevented a more specific route from reaching an edge router that performs accounting. In this case, the lack of proper tagging does not affect connectivity, but breaks billing for the network.

*Missing outgoing filter*. The local network policy treats a set of neighboring ASes differently and does not advertise certain routes to them. The outgoing route maps to these networks therefore typically start with the following:

```
route map to_abc deny 10
match community nwB_no_export
```

The route map to one of these special networks mistakenly lacks this match statement. One strength of Minerals is that it can detect such outliers without knowing the meaning of the statements involved. Minerals does not need to understand the function of the community-list `nwB_no_export`.

*Typos in community-list definitions*. In one router, there is a typo in a community-list definition. The local AS number is 123, and the typo is in one of the digits, making it 124:

```
community core_members [123:100 124:101]
```

*Distribution of internally originated routes*. One of the analyzed networks advertised its internal routes to a selected set of neighbors through a `match prefix-list internal-routes` command in which the prefix-list listed all the internal prefixes. This configuration is being changed and internal routes are now tagged with a specific community at origination. The configurations are updated to include a `match community` command to allow the propagation of internal routes to the desired parties. We found a number of BGP sessions still having the initial configurations. When `match community` is accidentally omitted in some of the sessions, some internal routes may be missing from the route advertisements. More specifically, internal routes not listed in the outdated prefix-list cannot be forwarded to the neighbors.

*Transit service*. In one network, Minerals found a research peer is receiving transit service for a wide range of prefixes. Usually, only customer routes are advertised to peers. This configuration does not violate the local network policies as it is a special case arrangement for a certain type of peers, but it is not a permanent situation.

### C. False Alarms

Most of the false alarms in the BGP instance type mainly consist of highly customized routing policies. In one network, there are a number of neighbor ASes that are multi-homed, only wanting to accept certain routes. Knowing that the configurations to these neighbors are unique and significantly different from the rest of the configurations, we could have removed them from our evaluation. However, we included them. A number of errors are detected on the sessions to these ASes, such as accidental omission of prefix-lists.

The false alarms in the user-account part of the configuration consisted of a number of special accounts created for some customers. These accounts presented different privilege levels than the other existing accounts and were therefore highlighted.

## VI. Limitations of Minerals

Minerals is simple to use, does not require any *a priori* model, and can detect violations of local network-specific policies without the need for customization. The scope of errors that can be detected is not restricted by predefined rules. We believe that Minerals could work in conjunction with the existing rules-based solutions to enhance misconfiguration detection. While existing methods are very efficient in identifying design errors, Minerals can highlight misconfigurations that would be difficult to pinpoint with a rules-based solution. For example, Minerals can identify the violations of the following policies: user $X$ is limited to privilege $k$; routes with communities $Y$ must be filtered; routes with communities $Z$ must be assigned a local preference $N$; etc. The detection of these errors through rules-based methods would require an intensive customization since the community values, the local preferences, and the usernames are network-specific.

However, Minerals also presents a number of limitations.

- *Error types*. It cannot detect errors in parameters that are specific and do not form a pattern. As an example, border routers connected to customers often apply prefix-lists and access-lists to limit the received signaling and traffic. Because each router may be connected to a different customer, Minerals cannot validate the correctness of the listed IP addresses. Also, Minerals may not find certain design errors (e.g., BGP sessions not forming a full mesh). Rules-based solutions are better suited to detect this latter category of errors.
- *Parser*. Minerals first needs to parse the configurations files. However, there are different versions of the operating systems (OS) and each version presents its own syntax. Therefore, the parser needs to cover the different possible grammars of the OS versions present in the analyzed network. This issue is not specific to our proposed approach, but is present in all methods that rely on the analysis of configuration files. We believe that this issue can be resolved in the future through the support of management models that offer a vendor independent representation of the configurations (e.g., [14]). It is important to note that even though the proposed approach relies on parsed data, it can still detect syntax errors as illustrated by the results in Section V. More specifically, the error must be parsed unchanged into the intermediate representation and be an anomaly out of a pattern.
- *Thresholds*. The proposed approach relies on a number of thresholds (e.g., $min\_conf$, $min\_supp$) which affect the accuracy of the results. A low $min\_conf$ increases the number of false alarms, while a high $min\_conf$ may miss errors. As an example, in one network, a user was limited to a specific privilege level. A fraction of its accounts were mistakenly assigned a higher privilege, but because the number of misconfigured accounts was large, the policy was not identified and the violation not raised. We proposed a number of heuristics to determine the thresholds, and future research can investigate additional methods.
- *Identification and definition of attributes*. The selection and definition of the attributes determines the quality of the re-

sults. Whether the value of an attribute should be taken as is, abstracted into a group, or converted into a Boolean determines the types of errors that can be detected. These decision steps strongly rely on the domain knowledge. However, the selection and definition of the attributes only need to be done once by the implementer of Minerals who possesses domain knowledge of router and network configurations, and it can then be used for all networks.

## VII. Discussion and Future Work

Minerals was successful in detecting a number of errors that would have been missed by other techniques. The operators who experimented with it found Minerals helpful and provided positive feedback. The extension of the model with additional attributes can help to unearth further mistakes. The analysis of statistical properties of router configurations appears to be a promising approach to assist operators in detecting mistakes.

We are exploring adding attributes and extending the number of instance types to broaden the scope of errors and areas where Minerals can be applied.

The analysis of the configurations over the temporal dimension could possibly reveal additional misconfigurations not detected by snapshot analysis. We have started to analyze successive snapshots of the configurations to expose patterns of change. We took monthly snapshots of the DC and HPR configurations from June 2004 through March 2006. We analyzed the evolution of user accounts on the routers and found that a handful of accounts are rotated regularly: a username would be added to almost all routers at about the same time, then deleted some time later, and replaced by another username. This turns out to be backdoor accounts that are created to ensure management access during times of failure, DoS, or planned maintenance events. Data mining can be applied to discover outliers in this pattern: e.g., routers that are misconfigured during rotation which either have multiple backdoor accounts, or new routers that are overlooked and have no backdoor accounts. An analysis over the temporal dimension can pick out these anomalies.

## Acknowledgment

I. Suilaiman, S. Rayanchu and S. Kim contributed to the implementation of the configuration parser. The authors thank Yongwon Lee for all his comments and suggestions. They would also like to express their gratitude to K. Lindahl, G. Kaplan, and others at CNS for providing access to Berkeley's configuration files, for suggestions and for education about network operations. Finally, they thank the anonymous reviewers and the editor for all their comments.

## References

[1] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.
[2] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. NSDI*, Boston, MA, May 2005, online.
[3] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002, pp. 3–16.
[4] B. J. P. Alin, C. Popescu, and T. Underwood, "Anatomy of a leak: AS9121 (or, "How we learned to start worrying and hate maximum prefix limits")," presented at the NANOG34 Meeting, Seattle, WA, May 2005.
[5] A. Feldmann and J. Rexford, "IP network configuration for intradomain traffic engineering," *IEEE Network*, vol. 15, no. 5, pp. 46–57, Sep./Oct. 2001.
[6] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting EDGE of IP router configuration," presented at the ACM SIGCOMM HotNets-II Workshop, Cambridge, MA, Nov. 2003.
[7] The Router Audit Tool (RAT). [Online]. Available: http://www.cisecurity.org/bench cisco.html
[8] K. El-Arini and K. Killourhy, "Bayesian detection of router configuration anomalies," presented at the ACM SIGCOMM Workshop on Mining Network Data (MineNet'05), Philadelphia, PA, Aug. 2005.
[9] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, Portland, OR, Aug. 2004, pp. 27–40.
[10] *"Router Security Configuration Guide System and Network Attack Center"* National Security Agency, 2003 [Online]. Available: http://www.nsa.gov/snac/routers/cisco scg-1.1b.pdf
[11] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford, "On static reachability analysis of IP networks," in *Proc. IEEE INFOCOM*, Miami, FL, May 2005, pp. 2170–2183.
[12] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Washington, DC, May 1993, pp. 207–216.
[13] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver.2: Efficient mining algorithms for frequent/closed/maximal itemsets," presented at the IEEE Int. Conf. Data Mining (ICDM'04) Workshop on Frequent Itemset Mining Implementations (FIMI'04), Brighton, U.K., Nov. 2004.
[14] *Common Information Model (CIM) Standards*, Distributed Management Task Force, Inc. [Online]. Available: http://www.dmtf.org/standards/cim/
[15] Cisco Netsys Connectivity Service Manager. Cisco, San Jose, CA [Online]. Available: www.cisco.com
[16] D. Engler, D. Y. Chen, and A. Chou, "Bugs as inconsistent behavior: A general approach to inferring errors in systems code," presented at the 18th ACM Symp. Operating Systems Principles (SOSP'01), Banff, Canada, Oct. 2001.
[17] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000, pp. 519–528.

**Franck Le** received the Diplome d'Ingenieur from the Ecole Nationale Superieure des Telecommunications de Bretagne, France, in 2000. He is currently working toward the Ph.D. degree at the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA.

From 2000 to 2005, he worked as a Research Engineer in the Mobile Networks Laboratory at Nokia Research Center, Irving, TX.

Mr. Le received the National Science Foundation Graduate Research Fellowship Award in 2006.

**Sihyung Lee** (S'08) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2000 and 2004, respectively. He is currently working toward the Ph.D. degree at Carnegie Mellon University, Pittsburgh, PA.

He is currently working on simplifying network management. His research interests include Internet routing, routing security, network management and measurement. His work is supported by the Samsung Scholarship Foundation.

**Tina Wong** received the B.S. degree (1995) with distinction in computer science from the University of Washington, Seattle, in 1995, and the M.S. and Ph.D. degrees in computer science from the University of California at Berkeley in 1998 and 2000, respectively.

She is a faculty member in the Information Networking Institute and a System Scientist at CyLab and ECE, all part of Carnegie Mellon University, Pittsburgh, PA. From 2000 to 2002, she was a Research Scientist at Hewlett Packard Laboratories, and worked on a collaboration with NTT DoCoMo on streaming media services for next-generation mobile devices. From 2002 to 2004, she was a member of the Network Science Department at Packet Design, Palo Alto, CA, during which she worked with a stellar group of people on various aspects of routing analytics, including a product. Her current research interests include simplifying network management, protecting the Internet routing infrastructure, and secure management of sensor networks.

**Hyong S. Kim** received the B.Eng. (Honours) degree in electrical engineering from McGill University in 1984, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1987 and 1990, respectively.

Since 1990, he has been with Carnegie Mellon University (CMU), Pittsburgh, PA, where he is currently the Drew D. Perkins Chaired Professor of Electrical and Computer Engineering. His primary research areas are advanced switching architectures, fault-tolerant, reliable, and secure network architectures, and optical networks. His Tera ATM switch architecture, developed at CMU, has been licensed for commercialization. In 1995, he founded Scalable Networks, a Gigabit-Ethernet switching startup. Scalable Networks was acquired by FORE Systems in 1996. In 2000, he founded AcceLight Networks, an optical switching startup, and was CEO until 2002. He is an author of over 70 published papers and holds more than 10 patents in networking technologies.

Dr. Kim was an editor for the IEEE/ACM TRANSACTIONS ON NETWORKING from 1995 to 2000. He was the recipient of the National Science Foundation Young Investigator Award in 1995.

**Darrell Newcomb** received the B.S. degree (1999) in information technology from the Rochester Institute of Technology, Rochester, NY, in 1999.

He is a Network Engineer with the Corporation for Education Network Initiatives in California (CENIC). He is currently providing technical leadership for the Transit Rail and CalREN networks from CENIC's offices in Cypress, CA. From 1999 to 2000, he was a Network Architect with NTT Multimedia Communications Laboratories, and was with Virage from 2000 to 2002.